1999

Computational complexity of term-equivalence

Clifford Bergman
Iowa State University, cbergman@iastate.edu

David Juedes
Ohio University

Giora Slutzki
Iowa State University, slutzki@iastate.edu

Computational complexity of term-equivalence

Abstract

Two algebraic structures with the same universe are called term-equivalent if they have the same clone of term operations. We show that the problem of determining whether two finite algebras of finite similarity type are term-equivalent is complete for deterministic exponential time.

Disciplines

Algebra | Numerical Analysis and Scientific Computing

Comments

This is a manuscript of an article published as Bergman, Clifford, David Juedes, and Giora Slutzki. "Computational complexity of term-equivalence." *International Journal of Algebra and Computation* 9, no. 01 (1999): 113-128. doi: 10.1142/S0218196799000084. Posted with permission.

COMPUTATIONAL COMPLEXITY OF TERM-EQUIVALENCE

CLIFFORD BERGMAN, DAVID JUEDES, AND GIORA SLUTZKI

ABSTRACT. Two algebraic structures with the same universe are called term-equivalent if they have the same clone of term operations. We show that the problem of determining whether two finite algebras of finite similarity type are term-equivalent is complete for deterministic exponential time.

When are two algebraic structures (presumably of different similarity types) considered to be "the same", for all intents and purposes? This is the notion of 'term-equivalence', a cornerstone of universal algebra.

For example, the two-element Boolean algebra is generally thought of as the structure $\mathbf{B} = \langle \{0,1\}, \wedge, \vee, \neg \rangle$, with the basic operations being conjunction, disjunction and negation. However, it is sometimes convenient to use instead the algebra $\mathbf{B}' = \langle \{0,1\}, \mid \rangle$ whose only basic operation is the Sheffer stroke. Algebraically speaking, these two structures behave in exactly the same way. It is easy to transform $\mathbf{B}$ into $\mathbf{B}'$ and back via the definitions

$$x \mid y = \neg(x \wedge y)$$

and

$$x \wedge y = (x \mid y) \mid (x \mid y), \quad x \vee y = (x \mid x) \mid (y \mid y), \quad \neg x = x \mid x.$$

Informally, two algebras $\mathbf{A}$ and $\mathbf{B}$ are called term-equivalent if each of the basic operations of $\mathbf{A}$ can be built from the basic operations of $\mathbf{B}$, and vice-versa. For a precise formulation, see Definition 3.3.

Because of its fundamental role, it is natural to wonder about the computational complexity of determining term-equivalence. Specifically, given two finite algebras $\mathbf{A}$ and $\mathbf{B}$ of finite similarity type, are $\mathbf{A}$ and $\mathbf{B}$ term-equivalent? There is a straightforward deterministic algorithm for solving this problem (Theorem 3.6), and its running time is exponential in the size of the input. In this paper we prove that this is the best possible: the term-equivalence problem is complete for **EXPTIME** (Theorem 3.8). It follows from the Hierarchy Theorem of Hartmanis and Stearns [5] that the class **EXPTIME** is strictly larger than **PTIME** (the class of problems solvable

Date: November 1997.

1991 *Mathematics Subject Classification.* Primary 08A40, 68Q15.

Key words and phrases. alternating Turing machine, tree automata, clone, term-equivalence, exponential time.

in polynomial time). See [6] or [12] for other proofs of the Hierarchy Theorem. Thus, a consequence of Theorem 3.8 is that the term-equivalence problem is not a member of **PTIME**. A problem is considered intractable if it fails to lie in **PTIME**.

This result has probably been known for quite some time. In 1977, D. Kozen [9] proved that if all of the basic operations of **A** and **B** are *unary,* then the problem of determining whether **A** and **B** are term-equivalent is complete for **PSPACE** (that is, the class of problems whose memory requirements are bounded by a polynomial in the size of the input). In a private communication, H. Friedman informed us that he discovered the general result about 1985, although it was never published.

Our proof involves two intermediate problems. The first utilizes the notion of a (deterministic) *bottom-up tree automaton* (Section 2). This is a generalization of a finite state machine which recognizes (finite) trees instead of strings. We shall introduce the problem $\textsc{Int-BTL}^1_r$, of determining whether a finite set of restricted bottom-up tree automata (of rank at most r—see Definition 2.1) recognize a common tree. By a restricted bottom-up tree automaton, we mean one with a unique final state. The second intermediate problem we call Gen-Clo. This is the determination of the clone generated by a family of operations on a finite set (see section 3).

The complete argument can be summarized as

$$\mathbf{EXPTIME} = \mathbf{APSPACE} \leq^p_m \textsc{Int-BTL}^1_2 \leq^p_m \textsc{Gen-Clo} \in \mathbf{EXPTIME},$$

$$\textsc{Gen-Clo} \leq^p_m \textsc{Term-Equiv} \in \mathbf{EXPTIME}.$$

Of course, Term-Equiv refers to the term-equivalence problem. The symbol '$\leq^p_m$' denotes polynomial-time, many-one reducibility, see any of [6], [10], [12] or [16]. The equality **EXPTIME** = **APSPACE** expresses the well-known result (see [2]) that the class **EXPTIME** is identical to the class of problems solvable by alternating Turing machines in polynomial space.

1. Alternating Turing machines

We assume that the reader is generally familiar with Turing machines and how they operate. We outline the specifics of our particular notation and conventions.

A single-tape (nondeterministic) Turing machine is a construct

$$\mathbf{M} = \langle\, Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R, \not{c} \,\rangle\,,$$

where Q is a set of *states,* Σ is the *input alphabet,* Γ is the *tape alphabet* ($\Sigma \subseteq \Gamma$), $q_0 \in Q$ is the *initial state of* $\mathbf{M}$, $q_A, q_R \in Q - \{q_0\}$ are the (unique) *accepting* and *rejecting* states, and $\not{c} \in \Gamma - \Sigma$ denotes the *blank* symbol. The *transition function* of $\mathbf{M}$ is

$$\delta\colon \big(Q - \{q_A, q_R\}\big) \times \Gamma \;\longrightarrow\; \mathcal{P}\big(Q \times (\Gamma - \{\not{c}\}) \times \{\mathsf{L}, \mathsf{R}, \mathsf{S}\}\big).$$

In other words, given a state q and a member of the tape alphabet a, $\delta(q, a)$ will be a set of triples of the form $\langle q', b, \mathsf{X}\rangle$. Each triple represents a possible action taken by $\mathbf{M}$ upon reading the symbol a while in the state q. The action $\langle q', b, \mathsf{X}\rangle$ means: write b on the tape, move to state q' and shift the tape head left, right or not at all, depending on the value of X. Without loss of generality, we assume that for all $q \in Q - \{q_A, q_R\}$ and for all $a \in \Gamma$, $|\delta(q, a)| = 2$. This just means that at any time in its computation, $\mathbf{M}$ has two "choices" of instructions unless $q = q_A$ or $q = q_R$, in which case $\mathbf{M}$ halts. Thus, q_A and q_R are *halting states.*

An *instantaneous description* (ID) of $\mathbf{M}$ is a string of the form $\alpha q \beta$ where $q \in Q$ and $\alpha\beta \in \Gamma^*$ is the tape content (omitting the blanks on either side); $\mathbf{M}$ is *scanning* the first symbol of β (or a blank symbol if β is the empty string). Intuitively, an ID gives a complete description of the current state of $\mathbf{M}$'s computation. For instance, the *initial* ID of $\mathbf{M}$ on input $w \in \Sigma^*$ is $I_0(w) = q_0 w$. An entire computation of $\mathbf{M}$ on w can be represented by a sequence of IDs: $I_0(w), I_1, I_2, \ldots, I_k$, where each I_{j+1} follows from I_j by the transition function.

A string w is *accepted* by $\mathbf{M}$ if there is some computation $I_0(w), I_1, \ldots, I_k$ in which I_k is an *accepting* ID, i.e., one of the form $\alpha q_A \beta$. $L(\mathbf{M})$ denotes the set of all strings accepted by $\mathbf{M}$. A string that is not accepted is said to be *rejected.*

The *size* of the ID $\alpha q \beta$ is the length $|\alpha\beta|$ of the string $\alpha\beta$. $\mathbf{M}$ is $p(n)$*-space bounded,* where p is a polynomial, if for every input $w \in \Sigma^*$ and for every ID I that is reachable from $I_0(w)$ by applying $\mathbf{M}$'s transition function, the size of I is bounded by $p(|w|)$. $\mathbf{M}$ is *polynomial-space bounded* if, for some polynomial $p(n) \geq n$, $\mathbf{M}$ is $p(n)$-space bounded. For a polynomial-space bounded Turing machine we shall assume, without loss of generality, that $\mathbf{M}$ eventually halts on every input: if some computation halts in state q_A, the input is accepted, and when all computations halt in state q_R, the input is rejected. The class of all languages definable by polynomial-space bounded Turing machines is denoted by **PSPACE**.

We next define *alternating Turing machines*, introduced in [2]. See also [16] for an up-to-date discussion. Let

$$\mathbf{M} = \langle\, Q, U, \Sigma, \Gamma, \delta, q_0, q_A, q_R, \not{c}\,\rangle$$

be a Turing machine defined as above, except that a set $U \subseteq Q - \{q_A, q_R\}$ is designated as the set of *universal* states. States in $Q - U - \{q_A, q_R\}$ are *existential.* An ID whose state is universal (existential) is called a *universal (existential)* ID.

The semantics of acceptance for alternating Turing machines generalizes the acceptance concept as defined above for (nondeterministic) Turing machines. Let $\mathbf{M}$ be a $p(n)$-space bounded alternating Turing machine and let $w \in \Sigma^*$. A *computation tree* of $\mathbf{M}$ on w is a *finite* (binary) tree t whose nodes are labeled by IDs of $\mathbf{M}$ such that the following properties hold:

- All of the IDs labeling the nodes of t have the same length, $p(|w|) + 1$; shorter IDs are padded with $\not{c}$.

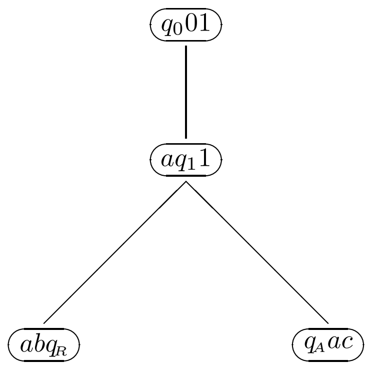

FIGURE 1

- The root of t is labeled with the initial ID of $\mathbf{M}$ on w (padded appropriately): $I_0(w) = q_0 w \not{c}^{p(|w|)-|w|}$.
- Let n be a node of t labeled by a universal ID I. Then n has two children in t, labeled by IDs I_0 and I_1 such that I_0 and I_1 follow from I according to the transition function δ of $\mathbf{M}$.
- Let n be a node of t labeled by an existential ID I. Then n has one child in t, labeled by ID J, such that J follows from I according to the transition function δ of $\mathbf{M}$.

Note that the IDs labeling the leaves of t are either accepting or rejecting IDs. We say that t is an *accepting computation tree* if all of the leaves of t are labeled by accepting IDs. The set of all accepting computation trees of $\mathbf{M}$ on w is denoted by $\mathrm{ACT}_{\mathbf{M}}(w)$. A string $w \in \Sigma^*$ is *accepted* by $\mathbf{M}$ if $\mathrm{ACT}_{\mathbf{M}}(w) \neq \emptyset$, i.e., if there exists an accepting computation tree of $\mathbf{M}$ on w. The language accepted by $\mathbf{M}$ is the set $L(\mathbf{M}) = \{ w : \mathrm{ACT}_{\mathbf{M}}(w) \neq \emptyset \}$. The class of all languages defined by polynomial-space bounded alternating Turing machines is denoted by **APSPACE**.

Figure 1 illustrates a very small computation tree t. Here, $w = 01$ and $p(n) = n$. The initial state q_0 is existential and q_1 is universal. The transition function δ satisfies $\langle q_1, a, \mathsf{R}\rangle \in \delta(q_0, 0)$ and $\{ \langle q_R, b, \mathsf{R}\rangle, \langle q_A, c, \mathsf{L}\rangle \} = \delta(q_1, 1)$. We have $t \notin \mathrm{ACT}_{\mathbf{M}}(w)$ since the left-hand leaf is a rejecting ID.

The standard (nondeterministic) model of Turing machines is obtained as a special case of the alternating model by taking $U = \emptyset$, i.e., all states are existential. It follows that $\mathbf{PSPACE} \subseteq \mathbf{APSPACE}$. In [2], Chandra, Kozen and Stockmeyer study the effect of introducing alternation on the complexity of the associated languages. In particular, they show that $\mathbf{APSPACE} = \mathbf{EXPTIME}$ where **EXPTIME** is defined to be $\mathbf{DTIME}(2^{\mathrm{poly}})$, the class of all languages defined by deterministic Turing machines in exponential time (the exponent being a polynomial in the input size). Thus one can show that an arbitrary problem P is hard for **EXPTIME** by starting with a polynomial-space bounded alternating Turing machine $\mathbf{M}$, and reducing the instance $\mathrm{ACT}_{\mathbf{M}}(w) \neq \emptyset$ to an instance of P. This is what we shall do in the next section.

2. Bottom-up tree automata

Here we give a brief description of the aspects of finite-state tree automata that we need. For a thorough treatment, see [3] or [4]. The latter reference has an extensive bibliography and historical comments.

That algebraic expressions can be represented as trees is surely familiar to the reader. For example, the Boolean algebra term $a \wedge (\neg(b \vee c))$ can be represented by the following tree.

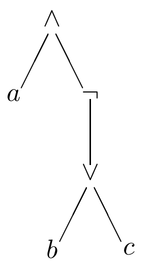

Figure 2

Note that in this tree, the leaves are labeled by symbols denoting elements, and the internal nodes by basic operation symbols. The number of children of each internal node is equal to the rank (arity) of the corresponding operation. We formalize this idea in the following definition.

Definition 2.1. A *ranked alphabet* Σ is a disjoint union of a finite family $\{\,\Sigma_k : 0 \leq k \leq r\,\}$ of finite sets. We require that $\Sigma_0 \neq \emptyset$. The integer r is called the *rank* of Σ.

If Σ is a ranked alphabet, the set of Σ-*trees,* $\mathcal{T}_\Sigma$, is the smallest set X of finite sequences such that

i. $\Sigma_0 \subseteq X$;
ii. If $k \geq 1$, $\sigma \in \Sigma_k$ and $t_1, t_2, \ldots, t_k \in X$, then $\sigma t_1 t_2 \ldots t_k \in X$.

We generally envision the members of $\mathcal{T}_\Sigma$ graphically, with the root at the top, as we have done in the tree above.

Definition 2.2. A (deterministic) *bottom-up tree automaton* is a structure $\langle\, \Sigma, Q, Q^*, R\,\rangle$ in which Q is a finite set (the set of states), Σ is a ranked alphabet, $Q^* \subseteq Q$ (the set of final states) and $R\colon \bigcup_{k=0}^{r} \Sigma_k \times Q^k \to Q$ (the transition rules). The rank of the automaton is the same as the rank of the alphabet.

Let us take note of one feature of this definition. Recall that Σ_0 is nonempty and $Q^0 = \{\star\}$ is a single object (technically, $Q^0 = \{\emptyset\}$). For each $\alpha \in \Sigma_0$, $R(\alpha, \star) = q_\alpha$ is the *initial state associated with* α.

Let $\mathbf{M} = \langle\, \Sigma, Q, Q^*, R\,\rangle$ be a bottom-up tree automaton. We can define a function $\widehat{M}\colon \mathcal{T}_\Sigma \to Q$ recursively by:

i. $\alpha \in \Sigma_0 \implies \widehat{M}(\alpha) = R(\alpha, \star)$;
ii. For $k \geq 1$, $\sigma \in \Sigma_k$ and $t_1, t_2, \dots, t_k \in \mathcal{T}_\Sigma$,
$\widehat{M}(\sigma t_1 t_2 \dots t_k) = R(\sigma, \widehat{M}(t_1), \widehat{M}(t_2), \dots, \widehat{M}(t_k))$.

Intuitively, $\widehat{M}(t)$ is the state in which $\mathbf{M}$ finds itself after processing the tree t.

Finally, we define $L(\mathbf{M}) = \{\, t \in \mathcal{T}_\Sigma : \widehat{M}(t) \in Q^* \,\}$ to be the *tree language recognized by* $\mathbf{M}$. The class of languages recognized by bottom-up tree automata is called **RECOG**.

Continuing with the Boolean algebra example we began just before Definition 2.1, the appropriate ranked alphabet Σ has $\Sigma_0 = \{a, b, c\}$, $\Sigma_1 = \{\neg\}$, $\Sigma_2 = \{\wedge, \vee\}$ and $\Sigma_k = \emptyset$, for $k > 2$. Thus, our alphabet has rank equal to 2. Let $\mathbf{B}$ be a finite Boolean algebra, $\bar{a}, \bar{b}, \bar{c}$ members of B, and X a subset of B. We can build a bottom-up tree automaton $\mathbf{M}$ as follows. The set Q of states is taken to be B, and $Q^* = X$. The transition rules are obtained from the operations of the algebra. Thus $R(a, \star) = \bar{a}$, $R(b, \star) = \bar{b}$ and $R(c, \star) = \bar{c}$. For each pair of states q_1, q_2, $R(\wedge, q_1, q_2)$ is defined to be the meet (a.k.a. conjunction) of q_1 and q_2 in $\mathbf{B}$. Use similar definitions for the other two operations. Now the tree depicted in Figure 2 above will be accepted by our machine $\mathbf{M}$ if and only if, in the algebra $\mathbf{B}$ we have $\bar{a} \wedge (\neg(\bar{b} \vee \bar{c})) \in X$. More generally, an arbitrary tree will be accepted by $\mathbf{M}$ if and only if, the value of the corresponding Boolean term, evaluated at $\bar{a}, \bar{b}, \bar{c}$, lies in X.

For our purposes, it is convenient to work with *restricted* bottom-up tree automata. By this we simply mean a bottom-up tree automaton in which $Q^* = \{q^*\}$ is a singleton. As we explain in Section 4, this assumption has no effect on the complexity of the problems we consider in this paper.

We can now state one of our basic problems, the intersection problem for bottom-up tree languages.

Definition 2.3. Let r be a nonnegative integer.

- Int-BTL^1_r denotes the set of all finite sequences $\langle\, \mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_\ell\,\rangle$ such that $\mathbf{M}_1$, … ,$\mathbf{M}_\ell$ are restricted bottom-up tree automata on a common ranked alphabet of rank at most r and $\bigcap_{i=1}^{\ell} L(\mathbf{M}_i) \neq \emptyset$.
- Int-BTL$^1 = \bigcup_{r=1}^{\infty}$ Int-BTL^1_r.
- Similarly, Int-BTL$_r$ and Int-BTL denote the corresponding intersection problem for unrestricted bottom-up tree automata.

Note that Definition 2.3 defines an infinite ascending sequence of problems, parameterized by r. On the other hand, the length ℓ of any instance of one of these problems is not bounded in any way.

Given an alternating Turing machine $\mathbf{M}$ and an input w, we wish to represent $\mathrm{ACT}_{\mathbf{M}}(w)$ as a (formal) language of trees that can be "read" by bottom-up tree automata. To do that we first need to transform the trees in $\mathrm{ACT}_{\mathbf{M}}(w)$ to trees in which each node is labeled by a single symbol from some ranked alphabet. Roughly, we will "rotate" each of the IDs labeling the nodes of a computation tree 90° clockwise into a long "unary" path. More precisely, for a given $p(n)$-space bounded alternating Turing machine $\mathbf{M} = \langle Q, U, \Sigma, \Gamma, \delta, q_0, q_A, q_R, \not{c} \rangle$, we define a ranked alphabet $\Psi = Q \cup \Gamma \cup \{\#_\forall, \#_\exists, \#_A, \#_R\}$ with ranks:

$$\Psi_2 = \{\#_\forall\}, \qquad \Psi_1 = Q \cup \Gamma \cup \{\#_\exists\}, \qquad \Psi_0 = \{\#_A, \#_R\}.$$

Given a computation tree t of $\mathbf{M}$ on w, with $|w| = n$, we transform t into a "transposed" tree T_t over the ranked alphabet Ψ by transforming each ID $I = s_1 s_2 \dots s_{p(n)+1}$ labeling a node of t into an ID-*segment:*

$$\begin{tikzcd}[row sep=small]
s_1 \arrow[d, dash] \\
s_2 \arrow[d, dash] \\
\vdots \arrow[d, dash] \\
s_{p(n)+1} \arrow[d, dash] \\
\$
\end{tikzcd}$$

where \$ is $\#_\forall$ (respectively $\#_\exists$, $\#_A$, $\#_R$) depending on whether I is a universal (respectively existential, accepting, rejecting) ID. For example, for the tree t of Figure 1, T_t will look like Figure 2.

The following Proposition follows directly from the definitions.

Proposition 2.4. *Let $\mathbf{M}$ be a $p(n)$-space bounded alternating Turing machine, and $w \in \Sigma^*$. Then*

$$w \in L(\mathbf{M}) \iff \mathrm{ACT}_{\mathbf{M}}(w) \neq \emptyset \iff \{\, T_t : t \in \mathrm{ACT}_{\mathbf{M}}(w) \,\} \neq \emptyset.$$

We next describe a set (of polynomial cardinality) of bottom-up tree automata $\{\, \mathbf{B}_i : 0 \leq i < p(n) + 3 \,\}$, each with polynomially many states, such that

$$\{\, T_t : t \in \mathrm{ACT}_{\mathbf{M}}(w) \,\} = \bigcap_i L(\mathbf{B}_i).$$

By Proposition 2.4, this implies that

$$w \in L(\mathbf{M}) \iff \bigcap_i L(\mathbf{B}_i) \neq \emptyset.$$

The construction of the $\mathbf{B}_i$'s is much like the construction of the finite-state automata F_i in [9]. Their objectives are the same: For $1 \leq i \leq p(n)$, $\mathbf{B}_i$ checks that for each ID-segment I of T_t, the 3 symbols beginning with the ith symbol:

$$\sigma_1 \sigma_2 \cdots \sigma_{i-1} \boxed{\sigma_i \sigma_{i+1} \sigma_{i+2}} \sigma_{i+3} \cdots \sigma_{p(n)+1} \$$$

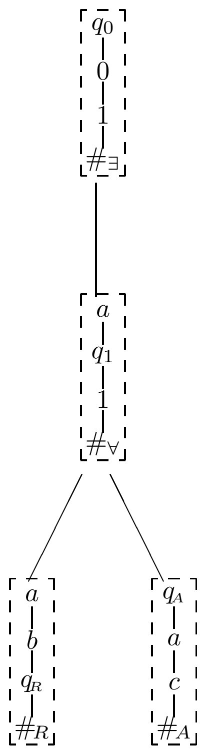

FIGURE 3

are consistent with the corresponding symbols in the ID-segment of the child node(s) according to the transition function δ of $\mathbf{M}$. Since each node of T_t has at most two children, $\mathbf{B}_i$ needs to check the consistency of a total of (at most) nine symbols of Ψ; thus, a straightforward construction of $\mathbf{B}_i$ gives $O\big(|\Psi|^9\, p(n)\big)$ states. There are three remaining bottom-up tree automata: $\mathbf{B}_0$ checks that the ID-segment starting at the root of T_t corresponds to the initial ID of $\mathbf{M}$ on w, and that the leaves of T_t are all labeled by $\#_A$; $\mathbf{B}_{p(n)+1}$ checks that the symbols $\#_A$, $\#_R$, $\#_\exists$ and $\#_\forall$ fit their corresponding ID-segments ($\#_A$ follows an accepting ID, etc.); and $\mathbf{B}_{p(n)+2}$ checks that each ID-segment has length $p(n)+2$.

We have constructed a total of $p(n)+3$ restricted bottom-up tree automata, each with $O\big(|\Psi|^9\, p(n)\big)$ states. The common alphabet, Ψ, has rank 2. It is easy to see that the construction of the bottom-up tree automata $\{\mathbf{B}_i\}$ from a given $p(n)$-space bounded alternating Turing machine $\mathbf{M}$ and input $w \in \Sigma^*$ can be done in polynomial time. Combining with Proposition 2.4 and our earlier observations, we have proved the following Theorem.

Theorem 2.5. *The problem* INT-BTL$^1_2$ *is hard for* **EXPTIME**.

3. Clones

We now develop the universal algebra needed to discuss term-equivalence. See [11] for a detailed treatment of the subject.

Let A be a set, k a positive integer. A *k-ary operation on A* is a function from A^k to A. We let $\mathrm{Op}_k(A)$ denote the set of all k-ary operations on A and $\mathrm{Op}(A) = \bigcup_{k=1}^{\infty} \mathrm{Op}_k(A)$ the set of all operations on A. An *algebra* is a pair $\mathbf{A} = \langle A, \mathcal{F} \rangle$, in which A is a nonempty set and $\mathcal{F} \subseteq \mathrm{Op}(A)$. The algebra $\mathbf{A}$ is of *finite similarity type* if the set $\mathcal{F}$ of basic operations is finite.

Let $f \in \mathrm{Op}_n(A)$ and $g_1, g_2, \dots, g_n \in \mathrm{Op}_k(A)$. The *generalized composition of f with $g_1, \dots, g_n$*, denoted $f[g_1, \dots, g_n]$, is the k-ary operation mapping $\mathbf{x} = (x_1, x_2, \dots, x_k)$ to $f\big(g_1(\mathbf{x}), g_2(\mathbf{x}), \dots, g_n(\mathbf{x})\big)$. Also, for any positive integer k and for $1 \leq i \leq k$, we define the *i^{th} k-ary projection operation* to be $p_i^k(\mathbf{x}) = x_i$. In particular, $p_1^1(x) = x$.

Definition 3.1. A *clone* on A is a set $\mathcal{C}$ of operations on A containing all projection operations and closed under generalized composition.

The most complete reference on clones is [14]. More recent treatments can be found in [11, Chapter 4] and [13, Chapter 1]. Here, we assemble just the small amount of information we need.

It is easy to see that the set of clones on A is ordered by set-theoretic inclusion. The largest clone is $\mathrm{Op}(A)$ itself and the smallest is the clone consisting of all projections. Furthermore, the intersection of a family of clones on A is again a clone. Therefore, for any set $\mathcal{F}$ of operations on A, we can define the clone generated by $\mathcal{F}$ to be

$$\mathrm{Clo}^A(\mathcal{F}) = \bigcap \{\, \mathcal{C} \subseteq \mathrm{Op}(A) : \mathcal{F} \subseteq \mathcal{C} \text{ and } \mathcal{C} \text{ is a clone} \,\}. \tag{1}$$

For our purposes, a more explicit description of $\mathrm{Clo}^A(\mathcal{F})$ is desirable. This is accomplished by the following Theorem, see [11, Theorem 4.3].

Theorem 3.2. *Let $\mathcal{F}$ be a set of operations on a set A and let n be a positive integer. The set $\mathrm{Clo}_n^A(\mathcal{F})$ of n-ary members of $\mathrm{Clo}^A(\mathcal{F})$ is the smallest set X of $\mathrm{Op}_n(A)$ such that*

i. *$p_i^n \in X$, for $i = 1, 2, \dots, n$;*
ii. *If $f \in \mathcal{F}$ and f is k-ary and $g_1, g_2, \dots, g_k \in X$ then $f[g_1, \dots, g_k] \in X$.*

Speaking informally, $g \in \mathrm{Clo}_n^A(\mathcal{F})$ if and only if g can be obtained by generalized composition from the members of $\mathcal{F}$ and the n-ary projection operations.

Definition 3.3. Two algebras $\mathbf{A} = \langle A, \mathcal{F} \rangle$ and $\mathbf{B} = \langle A, \mathcal{G} \rangle$ are called *term-equivalent* if $\mathrm{Clo}^A(\mathcal{F}) = \mathrm{Clo}^A(\mathcal{G})$.

Note that the definition requires that term-equivalent algebras have the same universe. Let $\mathbf{A}$ and $\mathbf{B}$ be as in Definition 3.3. In light of equation (1), $\mathbf{A}$ will be term-equivalent to $\mathbf{B}$ if and only if the following pair of conditions holds:

$$(\forall f \in \mathcal{F})\ f \in \mathrm{Clo}^A(\mathcal{G}) \quad \text{and} \quad (\forall g \in \mathcal{G})\ g \in \mathrm{Clo}^A(\mathcal{F}). \tag{2}$$

This suggests that in order to determine term-equivalence, the crucial point is to be able to perform tests of the form $g \in \mathrm{Clo}^A(\mathcal{F})$.

Definition 3.4. 1. Gen-Clo denotes the set of triples $\langle A, \mathcal{F}, h \rangle$ such that A is a finite nonempty set, $\mathcal{F}$ is a finite set of operations on A, h is an operation on A, and $h \in \mathrm{Clo}^A(\mathcal{F})$.
2. Term-Equiv denotes the set of pairs $\langle \mathbf{A}, \mathbf{B} \rangle$ such that $\mathbf{A}$ and $\mathbf{B}$ are finite, term-equivalent algebras of finite similarity type.

Our plan is to first turn our attention to Gen-Clo and prove that it is complete for **EXPTIME**. Then we use this result to draw the same conclusion for Term-Equiv.

Perhaps it would be worthwhile to say a few words about the format we would anticipate for the input in Gen-Clo. The finite set A could be represented by its cardinality, say m. (And A could be taken to be $\{1, 2, \ldots, m\}$.) A k-ary operation can be represented as a k-dimensional array of elements of A. In practice, this is nothing but a sequence of m^k members of $\{1, 2, \ldots, m\}$.

Theorem 3.5. *For any positive integer r,* Int-BTL$^1_r \leq^p_m$ Gen-Clo.

Proof. Let $\langle \mathbf{M}_1, \ldots, \mathbf{M}_\ell \rangle$ be an instance of Int-BTL^1_r, where for each $i \leq \ell$, $\mathbf{M}_i = \langle \Sigma, Q_i, \{q_i^*\}, R_i \rangle$. Without loss of generality, assume that the sets Q_i are pairwise disjoint. We must build an instance of Gen-Clo.

Let $A = \bigcup_{i=1}^{\ell} Q_i \cup \{\diamondsuit\}$, where $\diamondsuit$ is a new element. First, for every $\alpha \in \Sigma_0$, we define a unary operation f_α by

$$f_\alpha(q) = \begin{cases} R_i(\alpha, \star) & \text{if } q \in Q_i \\ \diamondsuit & \text{if } q = \diamondsuit. \end{cases}$$

For every $1 \leq k \leq r$ and every $\sigma \in \Sigma_k$ we define a k-ary operation f_σ by

$$f_\sigma(q_1, q_2, \ldots, q_k) = \begin{cases} R_i(\sigma, q_1, \ldots, q_k) & \text{if } (\exists i \leq \ell)\ \{q_1, \ldots, q_k\} \subseteq Q_i \\ \diamondsuit & \text{otherwise.} \end{cases}$$

Finally, define a unary operation h by

$$h(q) = \begin{cases} q_i^* & \text{if } q \in Q_i \\ \diamondsuit & \text{if } q = \diamondsuit. \end{cases}$$

Let $\mathcal{F} = \{ f_\sigma : \sigma \in \Sigma \}$. We shall prove that $\bigcap L(\mathbf{M}_i)$ is nonempty just in case $h \in \mathrm{Clo}^A(\mathcal{F})$.

Let $t \in \mathcal{T}_\Sigma$. We can define, by recursion on the length of t, a unary operation f_t. If $t = \alpha \in \Sigma_0$, then f_t is already defined. Otherwise, for $t = \sigma t_1 t_2 \ldots t_k$ we set

$$f_t = f_\sigma[f_{t_1}, f_{t_2}, \ldots, f_{t_k}]. \tag{3}$$

It is easy to verify that if $q \in Q_i$ and $t \in \mathcal{T}_\Sigma$, then $f_t(q) = \widehat{M_i}(t)$. Therefore, for every $i \le \ell$,

$$\widehat{M_i}(t) = q_i^* \iff (\forall q \in Q_i)\ f_t(q) = q_i^*. \tag{4}$$

Claim. $\mathrm{Clo}_1^A(\mathcal{F}) = \{\, f_t : t \in \mathcal{T}_\Sigma \,\} \cup \{p_1^1\}$.

Proof of Claim. The right-to-left inclusion is obvious from equation (3). For the converse, we apply Theorem 3.2. For this, we need to check that $X = \{\, f_t : t \in \mathcal{T}_\Sigma \,\} \cup \{p_1^1\}$ satisfies the two conditions of Theorem 3.2, with $n = 1$.

Certainly $p_1^1 \in X$ by definition. If $f \in \mathcal{F}$ is k-ary and $g_1, \dots, g_k \in X$ then for some $\sigma \in \Sigma_k$ and $t_i \in \mathcal{T}_\Sigma$ $(i = 1, \dots, k)$, $f = f_\sigma$ and $g_i = f_{t_i}$. Consequently $f[g_1, \dots, g_k] = f_t \in X$ where $t = \sigma t_1 t_2 \dots t_k$. □

From the Claim, $h \in \mathrm{Clo}_1^A(\mathcal{F})$ if and only if there is a tree $t \in \mathcal{T}_\Sigma$ such that $h = f_t$. Using (4),

$$f_t = h \iff (\forall i \le \ell)\ \widehat{M_i}(t) = q_i^* \iff t \in \bigcap_{i=1}^{\ell} L(\mathbf{M}_i).$$

Thus we have a reduction of Int-BTL$_r^1$ to Gen-Clo. It remains to show that this reduction can be carried out in polynomial time. Actually, any individual component of $\mathcal{F}$ is easy to compute. The only issue is: how long will it take to write the output down? The amount of time it takes to do this is clearly linearly bounded by the size of the output. Thus, we wish to show that the size of $\langle A, \mathcal{F}, h\rangle$ is bounded above by a polynomial in the size of $\langle \mathbf{M}_1, \dots, \mathbf{M}_\ell\rangle$.

By adding some superfluous states, we can assume that $|Q_1| = |Q_2| = \cdots = |Q_\ell| = n$. Consider the size of $\mathbf{M}_i$. This is essentially equal to

$$|R_i| = \sum_{k=0}^{r} |\Sigma_k| \cdot n^k \cdot \log(n).$$

Therefore the total size of the input is

$$\sum_{i=1}^{\ell} |R_i| = \ell \cdot \sum_{k=0}^{r} |\Sigma_k|\, n^k \log(n) \ge \sum_{k=0}^{r} |\Sigma_k|\, \ell n^k. \tag{5}$$

Now for the output. $|A| = 1 + \sum_{i=1}^{\ell} |Q_i| \approx \ell n$. The total size of all operations in $\mathcal{F}$ is therefore

$$\sum_{k=0}^{r} |\Sigma_k|\, (\ell n)^k \log(\ell n) \le \sum_{k=0}^{r} |\Sigma_k|\, \ell^{k+1} n^{k+1} \le \Big(\sum_{k=0}^{r} |\Sigma_k|\, \ell n^k\Big)^{r+1}.$$

Combining these two inequalities, we see that the size of the output is bounded above by a polynomial (of degree $r+1$) in the size of the input. □

In order to complete the proof of the main result, we need to bound the complexity of Gen-Clo. We do that now.

Theorem 3.6. Gen-Clo $\in$ **EXPTIME**.

Proof. Let $\langle A, \mathcal{F}, h \rangle$ be an instance of Gen-Clo. Let $m = |A|$, n be the rank of h and let r be the maximum rank of any member of $\mathcal{F} \cup \{h\}$. Observe that $|\mathrm{Op}_n(A)| = m^{m^n}$.

To test whether h is a member of $\mathrm{Clo}_n^A(\mathcal{F})$, we can systematically construct the members of $\mathrm{Clo}_n^A(\mathcal{F})$ as follows. Let

$$X_0 = \{\, p_i^n : 1 \le i \le n \,\}$$

and for every natural number j

$$X_{j+1} = X_j \cup \{\, f[g_1, \dots, g_k] : f \in \mathcal{F}, f \text{ is } k\text{-ary and } g_1, \dots, g_k \in X_j \,\}\,.$$

It is easy to argue by induction that $X_0 \subseteq X_1 \subseteq X_2 \subseteq \cdots \subseteq \mathrm{Clo}_n^A(\mathcal{F}) \subseteq \mathrm{Op}_n(A)$. Since $|\mathrm{Op}_n(A)| = m^{m^n}$, there is an index $s \le m^{m^n}$ such that $X_s = X_{s+1}$. But the pair of conditions $X_s \supseteq X_0$ and $X_s = X_{s+1}$ are precisely the conditions of Theorem 3.2. Therefore $\mathrm{Clo}_n^A(\mathcal{F}) = X_s$. Thus we can solve this instance of the problem by constructing the sets X_j, for $j \le m^{m^n}$ and testing each one for the presence of h.

Now, how long will this computation take? First, for a fixed $f \in \mathcal{F}$ of rank k and $g_1, \dots, g_k \in X_j$, we can construct the table for $f[g_1, \dots, g_k]$ in time $O(m^n)$, since that is the size of the table. Since the maximum rank of any operation is r, we can compute X_{j+1} from X_j in time on the order of

$$|X_j|^r \cdot m^n \cdot |\mathcal{F}| \le \left(m^{m^n}\right)^r \cdot m^n \cdot |\mathcal{F}| \approx m^{(rm^n)}\, |\mathcal{F}|\,.$$

Call this last quantity T. We may have to do this for $j = 1, 2, \dots, m^{m^n}$, so the total running time is on the order of $m^{m^n}T$. Taking the log of this quantity (to the base 2) we obtain

$$(r+1)m^n \log(m) + \log(|\mathcal{F}|) \le (r+1)m^{n+1} + \log(|\mathcal{F}|). \tag{6}$$

To obtain an exponential upper bound, we need to argue that the quantity in expression (6) is bounded above by a polynomial in the size of the input. Assume that $m > 1$ and let S denote the size of $\langle A, \mathcal{F}, h\rangle$. Since $\mathcal{F}$ contains an r-ary operation, the size of $\mathcal{F}$ is at least $m^r \ge rm$. Also, the size of $\mathcal{F}$ is at least $|\mathcal{F}|$, since each operation contributes at least one to the overall size. The size of h is exactly m^n. Therefore, $S \ge m^n + rm$ and $S \ge |\mathcal{F}|$. If we take $p(S) = S^2 + S$, then

$$p(S) \ge (m^n + rm)^2 + |\mathcal{F}| \ge 2rm^{n+1} + |\mathcal{F}|$$

which dominates the right-hand side of inequality (6). Thus the algorithm runs in exponential time. □

Combining Theorems 2.5, 3.5 and 3.6 we have the following.

Theorem 3.7. Gen-Clo *and* Int-BTL$_r^1$ *(for* $r \ge 2$*) are complete for* **EXPTIME**.

Finally, we return to the primary interest in this paper.

Theorem 3.8. Term-Equiv *is complete for* **EXPTIME**.

Proof. Clearly, a triple $\langle A, \mathcal{F}, h\rangle$ is a member of Gen-Clo if and only if $\big\langle\, \langle A,\mathcal{F}\rangle,\ \langle A,\mathcal{F}\cup\{h\}\rangle\, \big\rangle \in$ Term-Equiv. Thus, Gen-Clo $\leq^p_m$ Term-Equiv.

On the other hand, Term-Equiv $\in$ **EXPTIME**. To see this, suppose we are given a pair $\langle \mathbf{A}, \mathbf{B}\rangle$, where $\mathbf{A} = \langle A, \mathcal{F}\rangle$ and $\mathbf{B} = \langle A, \mathcal{G}\rangle$ are algebras on the same universe. According to the assertion in (2), we can test $\langle \mathbf{A}, \mathbf{B}\rangle \in$ Term-Equiv by executing several (independent) calls to Gen-Clo. By Theorem 3.6 the running time of each such subroutine call is exponential in a polynomial in the size of one of the algebras. Therefore there is a polynomial p such that each call to Gen-Clo will run in time at most $2^{p(S)}$, where S is the size of $\langle \mathbf{A}, \mathbf{B}\rangle$. Since the number of calls is $|\mathcal{F}| + |\mathcal{G}| \leq S$, the entire computation runs in exponential time.

From Theorem 3.7 and the relationship

$$\text{Gen-Clo} \leq^p_m \text{Term-Equiv} \in \mathbf{EXPTIME},$$

we obtain the desired result. □

4. Remarks

A. It is interesting to compare Theorem 3.7 to several similar results involving the complexity of algebraic problems. The class of problems solvable by deterministic Turing machines in time that is bounded by a polynomial function in the size of the input is denoted **PTIME**. Similarly, **NLOGSPACE** is the class of problems solvable by nondeterministic Turing machines using space that is bounded by the logarithm of the input size.

The classes we have discussed in this paper are ordered linearly:

$$\begin{aligned}\mathbf{NLOGSPACE} \subseteq \mathbf{PTIME} \subseteq \mathbf{PSPACE} = \\ \mathbf{NPSPACE} \subseteq \mathbf{APSPACE} = \mathbf{EXPTIME}.\end{aligned}$$

The equality of **PSPACE** and **NPSPACE** is a result of Savitch [15]. See [12] or [16] for a complete treatment of the hierarchy of complexity classes.

Let $\mathbf{A}$ be an algebra and $\emptyset \neq X \subseteq A$. The *subalgebra generated by* X is

$$\mathrm{Sg}^{\mathbf{A}}(X) = \bigcap \{\, B : X \subseteq B \text{ and } \mathbf{B} \text{ is a subalgebra of } \mathbf{A} \,\}.$$

We introduce the following two problems.

Definition 4.1.
1. Gen-SubAlg is the set of triples $\langle \mathbf{A}, X, a\rangle$ such that $\mathbf{A}$ is a finite algebra of finite similarity type, $X \subseteq A$ and $a \in \mathrm{Sg}^{\mathbf{A}}(X)$.
2. Gen-SubSg is the subset of Gen-SubAlg in which $\mathbf{A}$ is a finite semigroup, i.e., $\mathbf{A} = \langle A, \cdot\rangle$, where '$\cdot$' is an associative binary operation.

In the literature, Gen-SubAlg has usually been referred to simply as Gen. Jones and Laaser [7] proved that Gen-SubAlg is complete for **PTIME**. On the other hand, Jones, Lien and Laaser [8] showed that Gen-SubSg is complete for **NLOGSPACE**.

Now let $\langle A, \mathcal{F}, h\rangle$ be an instance of Gen-Clo and let $\mathbf{A}$ be the algebra $\langle A, \mathcal{F}\rangle$. Theorem 3.2 can be interpreted as asserting that $\mathrm{Clo}^A_n(\mathcal{F})$ is the subalgebra of $\mathbf{A}^{|A|^n}$ generated by $\{\, p^n_i : 1 \leq i \leq n \,\}$. (Here, $\mathbf{A}^k$ refers to the

kth direct power of $\mathbf{A}$.) Thus every instance of Gen-Clo can be turned into an instance of Gen-SubAlg—but only by inflating the input from size $|A|^n$ to $|A|^{|A|^n} \approx 2^{|A|^{n+1}}$. In other words, Gen-Clo can be thought of as a special case of a "succinct" version of Gen-SubAlg.

A similar "exponential" relationship holds between the problem Gen-SubSg and the problem Gen$_1$-Clo:

$$\text{Gen}_1\text{-Clo} = \{\, \langle A, \mathcal{F}, h \rangle \in \text{Gen-Clo} : \mathcal{F} \cup \{h\} \subseteq \mathrm{Op}_1(A) \,\}\,.$$

In [9], Kozen showed that Gen$_1$-Clo is complete for **PSPACE**. Note that $\mathrm{Clo}_1^A(\mathcal{F})$ can be thought of as the subsemigroup of $\mathrm{Op}_1(A)$ (under ordinary function composition) generated by $\mathcal{F}$. We can summarize the relationships between these four problems with the table in Figure 4.

NLOGSPACE Gen-SubSg	**(N)PSPACE** Gen$_1$-Clo
PTIME Gen-SubAlg	**EXPTIME** Gen-Clo

Figure 4

Many subproblems of Gen-SubAlg have been studied. The problem remains complete for **PTIME**, even when restricted to triples $\langle \mathbf{A}, X, a \rangle$ in which X is a singleton set and $\mathbf{A}$ has a single commutative binary operation [1]. In the same paper, Barrington and McKenzie consider a sequence of generalizations of associativity and show that the complexity of the corresponding problems increases from **NLOGSPACE** to **PTIME** as the assumption gets weaker.

B. Theorem 3.7 asserts that for any fixed $r > 1$ the problem Int-BTL^1_r is complete for **EXPTIME**. It was necessary to fix r because the running time of our reduction in Theorem 3.5 is bounded by a polynomial of degree $r + 1$ in the size of the input instance. What about the unbounded problem Int-BTL^1? I.e., what is the complexity of the intersection problem when r is not fixed, but is part of the input instance? It is certainly still hard for **EXPTIME** (Theorem 2.5), but is it in **EXPTIME**? The answer is affirmative. The exponential time algorithm consists of two parts. Given an instance $\langle\, \mathbf{M}_1, \dots, \mathbf{M}_\ell \,\rangle$ over a common ranked alphabet Σ (of rank r)

1. Construct the direct product bottom-up tree automaton $\mathbf{M} = \mathbf{M}_1 \times \mathbf{M}_2 \times \cdots \times \mathbf{M}_\ell$ in a manner analogous to the direct product construction for finite-state automata (see [6]). We will have $L(\mathbf{M}) = \bigcap_{i=1}^{\ell} L(\mathbf{M}_i)$. This construction can be completed in exponential time.

2. Test whether $L(\mathbf{M}) = \emptyset$. This can be done in polynomial time (in the size of $\mathbf{M}$). One way to do this would be to create a context-free grammar that would generate the empty string precisely when $L(\mathbf{M})$ is nonempty. Another way would be to build an alternating log-space Turing machine that accepts (a description of) $\mathbf{M}$ if and only if $L(\mathbf{M})$ is nonempty.

C. Finally, let us briefly consider unrestricted bottom-up tree automata. Suppose $\mathbf{M}$ is such a machine, with $Q^* = \{q_1^*, \dots, q_t^*\}$. For each $j = 1, 2, \dots, t$, let $\mathbf{M}^{(j)}$ be the machine identical to $\mathbf{M}$ except that only q_j^* is considered to be a final state. Then $\mathbf{M}^{(j)}$ is a restricted bottom-up tree automaton in the sense of this paper. Clearly, we have $L(\mathbf{M}) = \bigcup_{j=1}^{t} L\big(\mathbf{M}^{(j)}\big)$.

Now consider a sequence $\langle \mathbf{M}_1, \dots, \mathbf{M}_\ell \rangle$ of unrestricted machines. Without loss of generality, assume that $|Q_i^*| = t$ for $i = 1, \dots, \ell$. Then

$$\bigcap_{i=1}^{\ell} L(\mathbf{M}_i) = \bigcap_{i=1}^{\ell} \bigcup_{j=1}^{t} L\big(\mathbf{M}_i^{(j)}\big) = \bigcup_{J} \bigcap_{i=1}^{\ell} L\big(\mathbf{M}_i^{(J_i)}\big).$$

In the final union, J ranges over all ℓ-tuples $(J_1, J_2, \dots, J_\ell)$ in $\{1, 2, \dots, t\}^\ell$. Thus we can determine whether $\mathbf{M}_1 \dots, \mathbf{M}_\ell$ accept a common tree by testing t^ℓ-many instances of Int-BTL^1. Since $t \leq |Q_1| = n$, and referring to inequality (5), we determine that t^ℓ is bounded above by an exponential in the size of the input instance. And, by Theorem 3.7, each subproblem is solvable in exponential time. Therefore the unrestricted problem Int-BTL lies in **EXPTIME**.

Acknowledgments. We would like to thank both Joel Berman and Sándor Vágvölgyi for helpful discussions on the subject of this paper. We also wish to thank the referee for a very detailed, helpful and extremely speedy review.

References

1. D. A. M. Barrington and P. McKenzie, *Oracle branching programs and logspace versus P*, Information and Computation **95** (1991), 96–115.
2. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, *Alternation*, Jour. ACM **28** (1981), 114–133.
3. J. Engelfriet, *Tree automata and tree grammars*, Tech. Report DAIMI FN-10, University of Aarhus, Denmark, 1975, Lecture Notes.
4. F. Gécesg and M. Steinby, *Tree automata*, Akadémiai Kiadó, Budapest, 1984.
5. J. Hartmanis and R. Stearns, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc. **117** (1965), 285–306.
6. J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading, MA, 1979.
7. N. D. Jones and W. T. Lasser, *Complete problems for deterministic polynomial time*, Theoretical Computer Science **3** (1977), 105–117.
8. N. D. Jones, Y. E. Lien, and W. T. Laaser, *New problems complete for nondeterministic polynomial log space*, Math. Systems Theory **10** (1976), 1–17.

9. D. Kozen, *Lower bounds for natural proof systems*, 18th Annual Symposium on Foundations of Computer Science (Providence, R.I., 1977), IEEE Comput. Soc., Long Beach, CA, 1977, pp. 254–266.
10. ______, *Automata and computability*, Springer-Verlag, New York, 1997.
11. R. McKenzie, G. McNulty, and W. Taylor, *Algebras, Lattices, Varieties volume I*, Wadsworth & Brooks/Cole, Belmont, CA, 1987.
12. C. H. Papadimitriou, *Computational complexity*, Addison-Wesley, Reading, MA, 1994.
13. N. Pippenger, *Theories of computability*, Cambridge University Press, Cambridge, UK, 1997.
14. R. Pöschel and L. A. Kalužnin, *Funktionen-und relationenalgebren*, Birkhäuser, Basel, 1979.
15. W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Computer and Systems Sciences **4** (1970), no. 2, 177–192.
16. M. Sipser, *Introduction to the theory of computation*, PWS Publishing Company, Boston, MA, 1997.

Department of Mathematics, Iowa State University, Ames, Iowa 50011
E-mail address: `cbergman@iastate.edu`

School of Electrical Engineering and Computer Science, Ohio University, Athens, Ohio 45701
E-mail address: `juedes@ohiou.edu`

Department of Computer Science, Iowa State University, Ames, Iowa 50011
E-mail address: `slutzki@cs.iastate.edu`